ARMY RESEARCH LABORATORY

Two-Dimensional Translations, Rotations, and Intersections Using C++

by Robert J. Yager

ARL-TN-539 **June 2013**

NOTICES

Disclaimers

Army Research Laboratory

Aberdeen Proving Ground, MD 21005-5066

ARL-TN-539 **June 2013**

Two-Dimensional Translations, Rotations, and Intersections Using C++

Robert J. Yager
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)* June 2013	**2. REPORT TYPE** Final	**3. DATES COVERED (From - To)** 20 March 2013
4. TITLE AND SUBTITLE Two-Dimensional Translations, Rotations, and Intersections Using C++		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Robert J. Yager		**5d. PROJECT NUMBER** AH80
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: RDRL-WML-A Aberdeen Proving Ground, MD 21005-5066		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TN-539
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Two-dimensional operations, such as rotations, translations, and intersections, are tools that are essential for many types of scientific modeling. However, the C++ programming language does not natively perform them.

15. SUBJECT TERMS
rotation, translation, intersection, 2-D, two dimensional, C++, operation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Robert J. Yager
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	20	**19b. TELEPHONE NUMBER** (*Include area code*) 410-278-6689

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

Acknowledgments

The author would like to thank Luke Strohm of the U.S. Army Research Laboratory's Weapons and Materials Research Directorate. Mr. Strohm provided technical and editorial recommendations that improved the quality of this report.

INTENTIONALLY LEFT BLANK.

1. Introduction

Two-dimensional (2-D) operations, such as rotations, translations, and intersections, are tools that are essential for many types of scientific modeling. However, the C++ programming language does not natively perform them. This report documents a set of functions, written in C++, that can be used to perform 2-D rotations, translations, and intersections. All of the functions have been grouped into the y2DOps namespace, which is summarized at the end of this report.

The functions that are presented in this report are special cases of more general three-dimensional (3-D) functions.[1] Compared to the 3-D functions, the 2-D functions provide simpler interfaces and faster calculations.

2. Translation of a Point in Space

2.1 Derivation

Let the position vector $\vec{p}$ represent an arbitrary point in a plane, where

$$\vec{p} = p_x\hat{x} + p_y\hat{y} . \tag{1}$$

Furthermore, let $\vec{d}$ represent a displacement vector, where

$$\vec{d} = d_x\hat{x} + d_y\hat{y} . \tag{2}$$

If $\vec{p}'$ is used to represent $\vec{p}$ after it has been translated, then

$$\vec{p}' = (p_x + d_x)\hat{x} + (p_y + d_y)\hat{y} . \tag{3}$$

2.2 C++ Implementation

Translate2D() Code

```
inline void Translate2D(//<=========================PERFORMS A 2D TRANSLATION
    double p[2],//<-------COORDINATES TO TRANSLATE (MODIFIED BY THIS FUNCTION)
    const double d[2]){//<--------------------------------DISPLACEMENT VECTOR
  p[0]+=d[0] , p[1]+=d[1];
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

[1]Yager, R. J. *Three-Dimensional Translations, Rotations, and Intersections Using C++*; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, 2013, in press.

Translate2D() Parameters

p — **p** is a two-element array that stores the position vector that is described by equation 1 (**p**=$\{p_x, p_y\}$). Note that **p** is modified by the Translate() function, as described by equation 3.

d — **d** is a two-element array that stores the displacement vector that is described by equation 2 (**d**=$\{d_x, d_y\}$). **d** determines the amount and direction by which **p** is translated.

Translate2D() Example

Figure 1 shows point $\vec{p}$ being translated to a new position ($\vec{p}\,'$) by displacement vector $\vec{d}$.

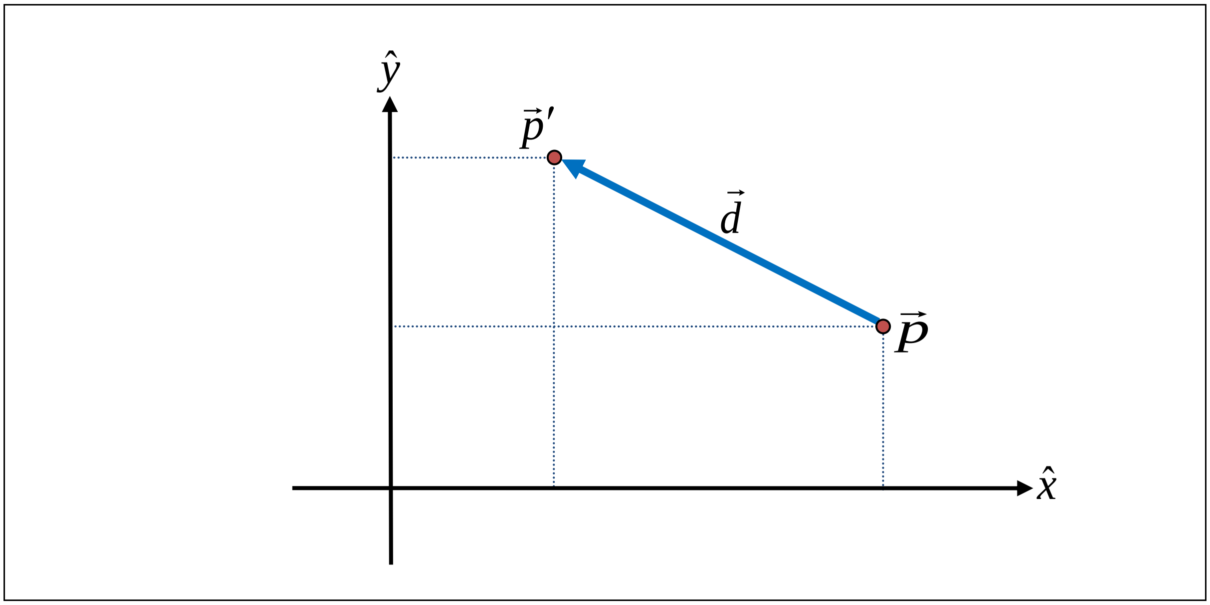


Figure 1. Translate() example.

Let $\vec{p} = \{3,1\}$ and $\vec{d} = \{-2,1\}$. Point $\vec{p}\,'$ can be found by using the Translate2D() function, as shown in the following sample code.

```
#include <cstdio>//.....................................................printf()
#include "y_2d_ops.h"//.............................................Translate2D()
int main(){
  double p[2]={3,1};
  double d[2]={-2,1};
  y2DOps::Translate2D(p,d);
  printf("p[0]=%f, p[1]=%f\n",p[0],p[1]);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

OUTPUT:

```
p[0]=1.000000, p[1]=2.000000
```

3. Rotation of a Point About an Arbitrarily Positioned Axis

3.1 Derivation

Suppose that the unit vector $\hat{v}$ is used to define an arbitrary axis about which a point in space will be rotated.

$$\hat{v} = v_x \hat{x} + v_y \hat{y} + v_z \hat{z}\,. \tag{4}$$

Rodrigues's rotation formula can be used to construct a rotation matrix,[2] R, that can be used to perform a rotation about $\hat{v}$ by an angle θ. The direction of the rotation can be determined by using the right-hand-thumb rule (when the right thumb is pointed in the direction of $\hat{v}$, the curled fingers of the right hand will point in the direction of the rotation).

$$R = \begin{bmatrix} v_x^2(1-c_\theta)+c_\theta & v_x v_y(1-c_\theta)-v_z s_\theta & v_x v_z(1-c_\theta)+v_y s_\theta \\ v_x v_y(1-c_\theta)+v_z s_\theta & v_y^2(1-c_\theta)+c_\theta & v_y v_z(1-c_\theta)-v_x s_\theta \\ v_x v_z(1-c_\theta)-v_y s_\theta & v_y v_z(1-c_\theta)+v_x s_\theta & v_z^2(1-c_\theta)+c_\theta \end{bmatrix}, \tag{5}$$

where

$$c_\theta \equiv \cos(\theta) \tag{6}$$

and

$$s_\theta \equiv \sin(\theta)\,. \tag{7}$$

For the 2-D case, assume that $\hat{v}$ points in the positive $\hat{z}$ direction. Then

$$v_x = 0\,,\ v_y = 0\,, \text{ and } v_z = 1\,. \tag{8}$$

This greatly simplifies equation 5:

$$R = \begin{bmatrix} c_\theta & -s_\theta & 0 \\ s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{9}$$

Substituting equations 6 and 7 into equation 5, then converting to 2D,

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{10}$$

---

[2] Mason, M. T. *Mechanics of Robotic Manipulation*; Massachusetts Institute of Technology Press: Cambridge, MA, 2001, (p 46, equation 3.26).

Let the position vector $\vec{p}$ locate an arbitrary point in a plane, where

$$\vec{p} = p_x\hat{x} + p_y\hat{y}\,. \tag{11}$$

Let the position vector $\vec{o}$ locate the origin of the rotation axis defined by $\hat{v}$.

$$\vec{o} = o_x\hat{x} + o_y\hat{y}\,. \tag{12}$$

The translation-rotation-translation sequence described by equation 13 can be used to find $\vec{p}\,'$, where $\vec{p}\,'$ is used to represent $\vec{p}$ after it has been rotated about $\hat{v}$.

$$\vec{p}\,' = R(\vec{p}-\vec{o})+\vec{o}\,. \tag{13}$$

3.2 C++ Implementation

Two functions are used to perform 2-D rotations. The first function, RMatrix2D(), calculates the rotation matrix that is presented in equation 10. The second function, Rotate2D(), performs the rotation that is presented in equation 13. Breaking the calculation into two functions allows functions that rotate objects containing more than one point to be written in a manner that doesn't sacrifice performance.

Rmatrix2D() Code

```
inline void RMatrix2D(//<======================CALCULATES A 2D ROTATION MATRIX
    double R[4],//<--------------ROTATION MATRIX (CALCULATED BY THIS FUNCTION)
    double rads){//<---------------THE ANGLE OF THE ROTATION (CCW IS POSITIVE)
  R[0]=R[3]=cos(rads) , R[1]=-(R[2]=sin(rads));
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

Rmatrix2D() Parameters

R — **R** is a four-element array that stores the rotation matrix that is described by equation 10 (**R**=$\{R_{0,0}, R_{0,1}, R_{1,0}, R_{1,1}\}$). Note that **R** is modified by the Rmatrix2D() function. **R** is intended to be used as the third argument of the Rotate2D() function.

rads — **rads** is used to represent the angle (in radians) of the rotation. The direction of the rotation is counterclockwise (see figure 2).

Rotate2D() Code

```
inline void Rotate2D(//<===================================PERFORMS A ROTATION
    double p[2],//<----------COORDINATES TO ROTATE (MODIFIED BY THIS FUNCTION)
    const double o[2],//<-------------------THE ORIGIN OF THE AXIS OF ROTATION
    const double R[4]){//<----------------A ROTATION MATRIX (FROM RMatrix2D())
  double t0=p[0]-o[0] , t1=p[1]-o[1];
  p[0]=R[0]*t0+R[1]*t1+o[0] , p[1]=R[2]*t0+R[3]*t1+o[1];
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

Rotate2D() Parameters

p — **p** is a two-element array that stores the position vector that is described by equation 11 (**p**=$\{p_x, p_y\}$). Note that **p** is modified by the Rotate2D() function, as described by equation 13.

o — **o** is a two-element array that stores the position vector that is described by equation 12 (**o**=$\{o_x, o_y\}$). **o** is the point about which **p** is rotated.

R — **R** is a rotation matrix that has been precalculated using the RMatrix2D() function.

Rotate2D() Example

Figure 2 shows point $\vec{p}$ being rotated about $\vec{o}$ to a new position ($\vec{p}\,'$).

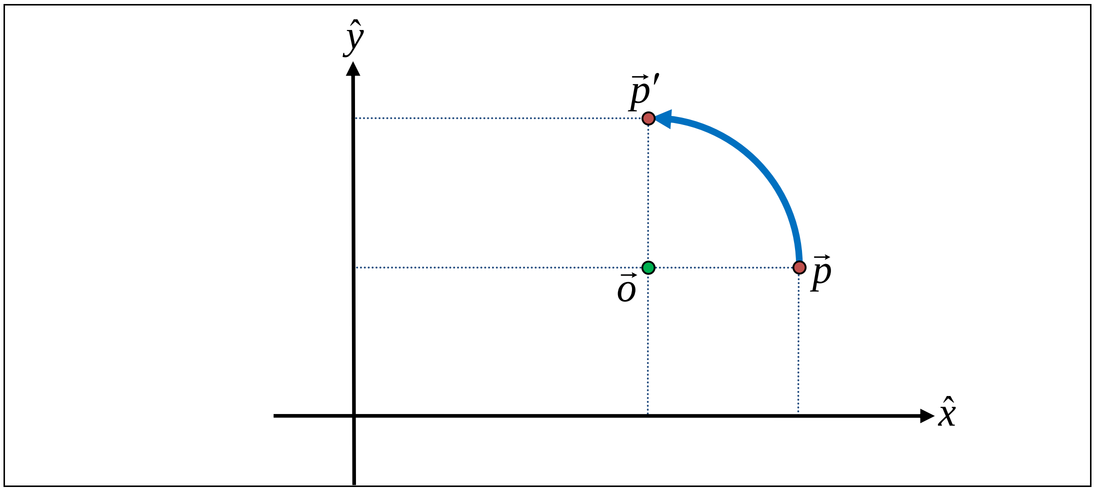


Figure 2. Rotate() example.

Let $\vec{p} = \{3,1\}$ and $\vec{o} = \{2,1\}$. Furthermore, let the angle of rotation be $\pi/2$. Point $\vec{p}\,'$ can be found by first using the RMatrix2D() function to calculate a rotation matrix, then using the Rotate2D() function to perform the rotation.

```
#include <cstdio>//......................................................printf()
#include "y_2d_ops.h
int main(){
  double p[2]={3,1};
  double o[2]={2,1};
  double R[4];/*<-*/y2DOps::RMatrix2D(R,3.14159265358979/2);//...rotation matrix
  y2DOps::Rotate2D(p,o,R);
  printf("p[0]=%f, p[1]=%f\n",p[0],p[1]);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

OUTPUT:

```
p[0]=2.000000, p[1]=2.000000
```

4. Intersection Between Two Lines

4.1 Derivation

Suppose that line A passes through the points $\vec{A}_0$ and $\vec{A}_1$ where

$$\vec{A}_0 = A_{0,x}\hat{x} + A_{0,y}\hat{y} \text{ and } \vec{A}_1 = A_{1,x}\hat{x} + A_{1,y}\hat{y}. \tag{14}$$

Let $\vec{p}_A$ represent a point that lies on A. $\vec{A}_0$ and $\vec{A}_1$ can be used to construct a parametric equation for $\vec{p}_A$ as a function of the parameter t_0:

$$\vec{p}_A = \vec{A}_0 + \left(\vec{A}_1 - \vec{A}_0\right)t_0. \tag{15}$$

The parameter t_0 represents the scaled distance from $\vec{A}_0$ to $\vec{A}_1$ along A. Thus, if $t_0 = 0$, $\vec{p}_A$ is located at $\vec{A}_0$. If $t_0 = 1$, $\vec{p}_A$ is located at $\vec{A}_1$.

Similarly, suppose that line B passes through the points $\vec{B}_0$ and $\vec{B}_1$ where

$$\vec{B}_0 = B_{0,x}\hat{x} + B_{0,y}\hat{y} \text{ and } \vec{B}_1 = B_{1,x}\hat{x} + B_{1,y}\hat{y}. \tag{16}$$

Let $\vec{p}_B$ represent a point that lies on B. $\vec{B}_0$ and $\vec{B}_1$ can be used to construct a parametric equation for $\vec{p}_B$ as a function of the parameter t_1:

$$\vec{p}_B = \vec{B}_0 + \left(\vec{B}_1 - \vec{B}_0\right)t_1. \tag{17}$$

Figure 3 presents an image of lines A and B for the case where they intersect.

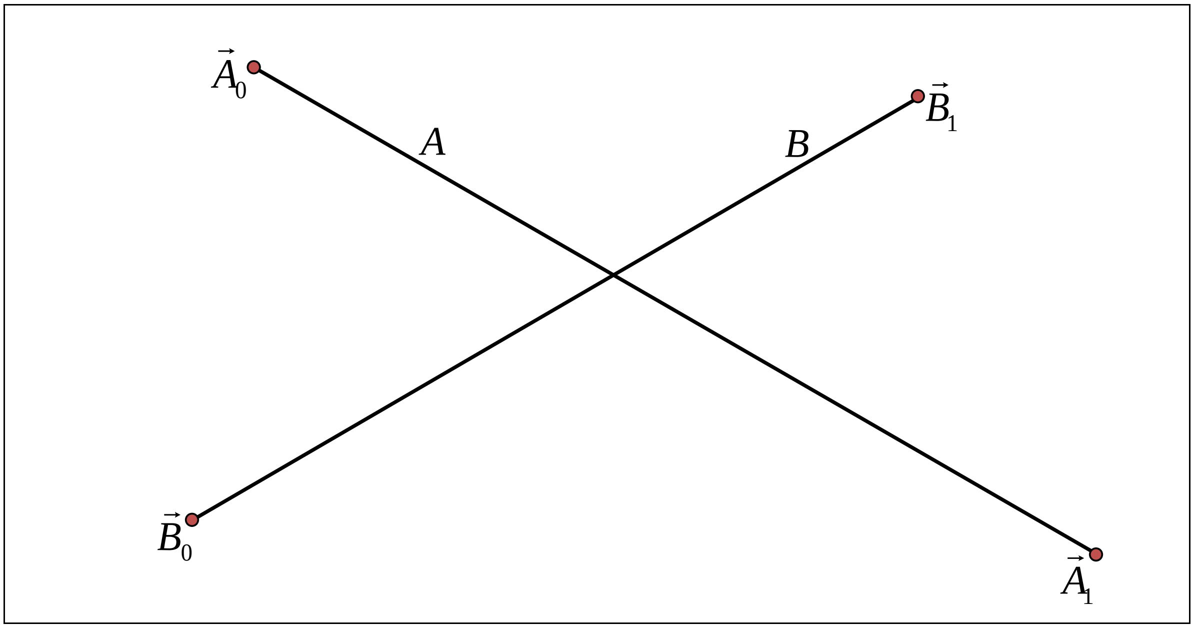


Figure 3. Intersecting lines *A* and *B*.

The point of intersection between A and B occurs where $\vec{p}_A$ is equal to $\vec{p}_B$. Thus,

$$\vec{A}_0 + \left(\vec{A}_1 - \vec{A}_0\right)t_0 = \vec{B}_0 + \left(\vec{B}_1 - \vec{B}_0\right)t_1 . \tag{18}$$

Rearranging terms,

$$\vec{A}_0 - \vec{B}_0 = \left(\vec{A}_0 - \vec{A}_1\right)t_0 + \left(\vec{B}_1 - \vec{B}_0\right)t_1 . \tag{19}$$

This can be written in matrix form as

$$\begin{bmatrix} A_{0,x} - B_{0,x} \\ A_{0,y} - B_{0,y} \end{bmatrix} = \begin{bmatrix} A_{0,x} - A_{1,x} & B_{1,x} - B_{0,x} \\ A_{0,y} - A_{1,y} & B_{1,y} - B_{0,y} \end{bmatrix} \begin{bmatrix} t_0 \\ t_1 \end{bmatrix} . \tag{20}$$

Solving for $\vec{t}$,

$$\begin{bmatrix} t_0 \\ t_1 \end{bmatrix} = \begin{bmatrix} A_{0,x} - A_{1,x} & B_{1,x} - B_{0,x} \\ A_{0,y} - A_{1,y} & B_{1,y} - B_{0,y} \end{bmatrix}^{-1} \begin{bmatrix} A_{0,x} - B_{0,x} \\ A_{0,y} - B_{0,y} \end{bmatrix} . \tag{21}$$

Recall that the vector $\vec{t}$ contains the parameters from equations 15 and 17. Thus, once $\vec{t}$ is known, t_0 can be substituted into equation 15 to find the point of intersection between A and B. Note that if the two-by-two matrix defined in equation 21 is noninvertible, then A is parallel to B.

Because t_0 and t_1 are defined to be scaled distances from $\vec{A}_0$ to $\vec{A}_1$ and $\vec{B}_0$ to $\vec{B}_1$, respectively, if the conditions presented in equation 22 are met, then the point of intersection lies between $\vec{A}_0$ and $\vec{A}_1$ and between $\vec{B}_0$ and $\vec{B}_1$, as shown in figure 3.

$$0 < t_1 < 1 \text{ and } 0 < t_2 < 1. \tag{22}$$

4.2 C++ Implementation

Two functions are used to find line-line intersections. The first function, IParameters2D(), calculates a two-element array that is the solution to equation 21. The second function, Intersect2D(), calculates the point of intersection between two lines.

Because there is a chance that the two-by-two matrix shown in equation 21 will be singular, a Boolean that indicates whether or not a solution is valid is returned by the IParameters2D() function.

IParameters2D() Code

```
inline bool IParameters2D(//<============PARAMETERS FOR LINE-LINE INTERSECTION
    double t[2],//<------INTERSECTION PARAMETERS (CALCULATED BY THIS FUNCTION)
    const double A[4],//<-----------------------------LINE A {A0X,A0Y,A1X,A1Y}
    const double B[4],//<-----------------------------LINE B {B0X,B)Y,B1X,B1Y}
    double e=1E-9){//<----CUTOFF VALUE FOR DETERMINING IF A AND B ARE PARALLEL
  double a=A[0]-A[2] , b=B[2]-B[0],//...............................2x2 matrix
         c=A[1]-A[3] , d=B[3]-B[1];//                                 elements
  double D=a*d-b*c;//...............................determinant of 2x2 matrix
  if(fabs(D)<e)return false;//....=> A & B are parallel (and t is meaningless)
  double f=A[0]-B[0] , g=A[1]-B[1];
  t[0]=(d*f-b*g)/D , t[1]=(-c*f+a*g)/D;
  return true;//....................................=> A & B are not parallel
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

IParameters2D() Parameters

R — **R** is a rotation matrix that has been precalculated using the RMatrix2D() function.

t — **t** is a two-element array that stores the parameters described in equation 21 (**t**=$\{t_0, t_1\}$). Note that **t** is modified by the IParameters2D() function.

A — **A** is a four-element array that stores the line that is defined by equation 14 (**A**=$\{A_{0,x}, A_{0,y}, A_{1,x}, A_{1,y}\}$).

B — **B** is a four-element array that stores the line that is defined by equation 16 (**B**=$\{B_{0,x}, B_{0,y}, B_{1,x}, B_{1,y}\}$).

e — **e** is the cutoff value for testing whether or not **A** and **B** are parallel. If the determinant of the matrix in equation 21 is less than **e**, then **A** and **B** are considered to be parallel. The default value of **e** is 10^{-9}.

IParameters2D() Return Value

IParameters2D() returns false if **A** is parallel to **B**. A return value of false indicates that **t** has not been calculated and, thus, should not be passed to the Intersect2D() function.

Intersect2D() Code

```
inline bool Intersect2D(//<============CALCULATES LINE-LINE INTERSECTION POINT
    double x[2],//<--------POINT OF INTERSECTION (CALCULATED BY THIS FUNCITON)
    const double t[2],//<-------INTERSECTION PARAMETERS (FROM IParameters2D())
    const double A[4]){//<---------------------------LINE A {A0X,A0Y,A1X,A1Y}
  x[0]=A[0]+t[0]*(A[2]-A[0]) , x[1]=A[1]+t[0]*(A[3]-A[1]);
  return t[0]>0&&t[0]<1&&t[1]>0&&t[1]<1;
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

Intersect2D() Parameters

x **x** is a two-element array that stores the point of intersection between lines **A** and **B**. Note that **x** is modified by the Intersect2D() function.

t **t** is a parameter list that has been precalculated using the IParameters2D() function.

A **A** is a two-element by two-element array that stores the line that is defined by equation 14 (**A**=$\{A_{0,x}, A_{0,y}, A_{1,x}, A_{1,y}\}$).

Intersect2D() Return Value

Intersect2D() returns true if line segment **A** intersects line segment **B**.

Intersect2D() Example

Figure 4 shows intersecting line-segments A and B.

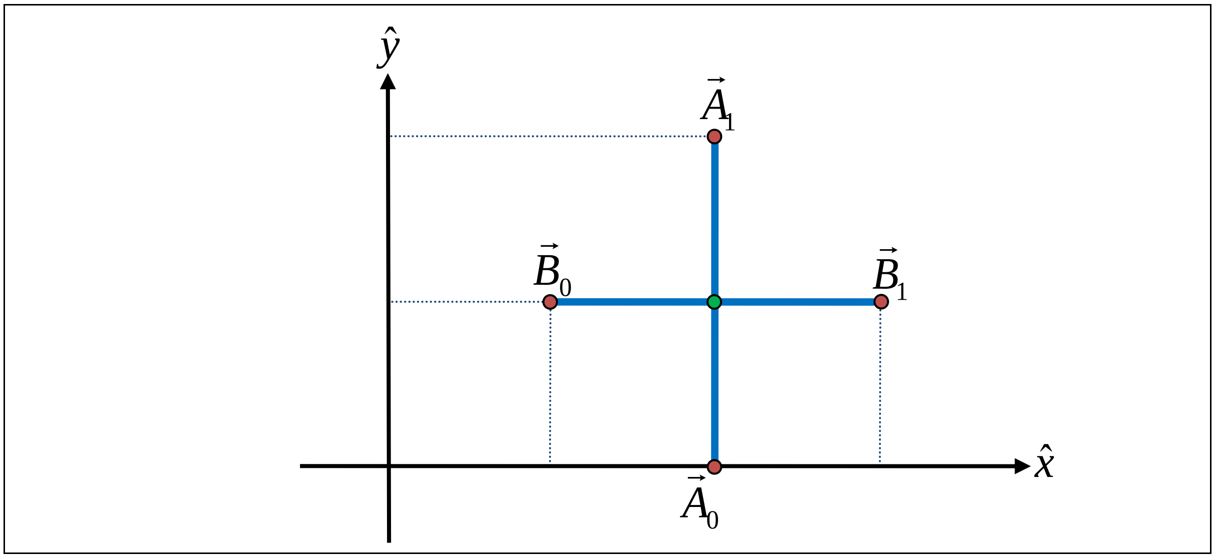


Figure 4. Intersect2D() example.

Let $\vec{A}_0 = \{2,0\}$, $\vec{A}_1 = \{2,2\}$, $\vec{B}_0 = \{1,1\}$, and $\vec{B}_1 = \{3,1\}$. The point of intersection can be found by first calling the IParameters2D() function, then using the result in the Intersect2D() function.

```
#include <cstdio>//.....................................................printf()
#include "y_2d_ops.h"
int main(){
  double A[4]={2,0 , 2,2};//......................................a line segment
  double B[4]={1,1 , 3,1};//......................................a line segment
  double t[2];/*<-*/y2DOps::IParameters2D(t,A,B);//......intersection parameters
  double p[2];/*<-*/y2DOps::Intersect2D(p,t,A);//..........point of intersection
  printf("p[0]=%f, p[1]=%f\n",p[0],p[1]);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

OUTPUT:

```
p[0]=2.000000, p[1]=1.000000
```

5. Summary

A summary sheet is provided at the end of this report. It presents the y2DOps namespace, which contains the five functions that are described in detail in sections 2, 3, and 4. Also presented are two examples that demonstrate the versatility of the functions described in this report. The first uses the Rotate2D() function to calculate a set of points that defines a simple orbit of a moon around a planet, which in turn is in orbit around a star. The second uses the Intersect2D() function to draw a four-sided spiral. Both functions create text files that contain all of the information needed to create the two images presented in the summary sheet.

Y2DOps Summary

y_2d_ops.h

```
#ifndef Y_2D_OPS_H_
#define Y_2D_OPS_H_
#include <cmath>//............................................sin(),cos(),abs()
namespace y2DOps{//...........TWO-DIMENSIONAL OPERATIONS (CARTESIAN COORDINATES)
  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  inline void Translate2D(//<===========================PERFORMS A 2D TRANSLATION
      double p[2],//<-------COORDINATES TO TRANSLATE (MODIFIED BY THIS FUNCTION)
      const double d[2]){//<---------------------------------DISPLACEMENT VECTOR
    p[0]+=d[0] , p[1]+=d[1];
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
  inline void Rotate2D(//<==================================PERFORMS A ROTATION
      double p[2],//<---------COORDINATES TO ROTATE (MODIFIED BY THIS FUNCTION)
      const double o[2],//<-------------------THE ORIGIN OF THE AXIS OF ROTATION
      const double R[4]){//<----------------A ROTATION MATRIX (FROM RMatrix2D())
    double t0=p[0]-o[0] , t1=p[1]-o[1];
    p[0]=R[0]*t0+R[1]*t1+o[0] , p[1]=R[2]*t0+R[3]*t1+o[1];
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
  inline void RMatrix2D(//<======================CALCULATES A 2D ROTATION MATRIX
      double R[4],//<--------------ROTATION MATRIX (CALCULATED BY THIS FUNCTION)
      double rads){//<---------------THE ANGLE OF THE ROTATION (CCW IS POSITIVE)
    R[0]=R[3]=cos(rads) , R[1]=-(R[2]=sin(rads));
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
  inline bool Intersect2D(//<============CALCULATES LINE-LINE INTERSECTION POINT
      double x[2],//<--------POINT OF INTERSECTION (CALCULATED BY THIS FUNCITON)
      const double t[2],//<-------INTERSECTION PARAMETERS (FROM IParameters2D())
      const double A[4]){//<---------------------------LINE A {A0X,A0Y,A1X,A1Y}
    x[0]=A[0]+t[0]*(A[2]-A[0]) , x[1]=A[1]+t[0]*(A[3]-A[1]);
    return t[0]>0&&t[0]<1&&t[1]>0&&t[1]<1;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
  inline bool IParameters2D(//<============PARAMETERS FOR LINE-LINE INTERSECTION
      double t[2],//<------INTERSECTION PARAMETERS (CALCULATED BY THIS FUNCTION)
      const double A[4],//<----------------------------LINE A {A0X,A0Y,A1X,A1Y}
      const double B[4],//<----------------------------LINE B {B0X,B)Y,B1X,B1Y}
      double e=1E-9){//<----CUTOFF VALUE FOR DETERMINING IF A AND B ARE PARALLEL
    double a=A[0]-A[2] , b=B[2]-B[0],//..............................2x2 matrix
           c=A[1]-A[3] , d=B[3]-B[1];//                                elements
    double D=a*d-b*c;//...............................determinant of 2x2 matrix
    if(fabs(D)<e)return false;//....=> A & B are parallel (and t is meaningless)
    double f=A[0]-B[0] , g=A[1]-B[1];
    t[0]=(d*f-b*g)/D , t[1]=(-c*f+a*g)/D;
    return true;//...................................=> A & B are not parallel
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
}//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
#endif
```

FIGURE 1

image created from orbit.txt.

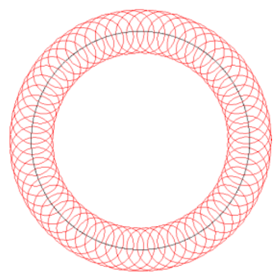
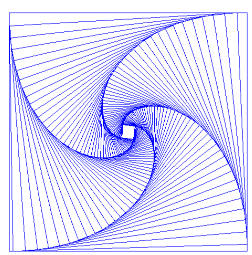


FIGURE 2

image created from spiral.txt



EXAMPLE 1

```
#include <cstdio>//.....................FILE,freopen(),stdout,printf(),fclose()
#include "y_2d_ops.h"//................................Rmatrix2D(),Rotate2D()
int main(){//<============================USE ROTATIONS TO MODEL CIRCULAR ORBITS
  using namespace y2DOps;
  FILE *f=freopen("orbit.txt","w",stdout);//..........redirect output to a file
  double s[2]={0,0},p[2]={1,0},m[2]={1.2,0};//.....sun, planet, & moon positions
  double Rp[4];/*<-*/RMatrix2D(Rp,.00436);//.........rotation matrix for planet
  double Rm[4];/*<-*/RMatrix2D(Rm,.279);//.............rotation matrix for moon
  printf("#   planet   |     moon\n#  x  ,  y   |   x  ,  y\n");//.column header
  for(int i=0;i<1444;++i){
    printf("%6.3f,%6.3f,%6.3f,%6.3f\n",p[0],p[1],m[0],m[1]);//..record positions
    Rotate2D(p,s,Rp),Rotate2D(m,s,Rp);//....rotate planet-moon system around sun
    Rotate2D(m,p,Rm);}//.............................rotate moon around planet
  fclose(f);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

orbit.txt

```
#   planet   |     moon
#  x  ,  y   |   x  ,  y
 1.000, 0.000, 1.200, 0.000
 1.000, 0.004, 1.192, 0.060
 1.000, 0.009, 1.169, 0.116
            .
            .
            .
```

EXAMPLE 2

```
#include <cstdio>//.....................FILE,freopen(),stdout,printf(),fclose()
#include "y_2d_ops.h"//...................<cmath>,IParameters2D(),Intersect2D()
int main(){//<======================USE LINE-LINE INTERSECTIONS TO MAKE A SPIRAL
  using namespace y2DOps;
  FILE *g=freopen("spiral.txt","w",stdout);//.........redirect output to a file
  double X[250][2]={ {-1,1} , {-1,-1} , {1,-1} , {1,1} };
  for(int i=3;i<249;++i){
    double a=(1.585)*(i-1)-.0285;//..........."a" increments by about 90 degrees
    double A[4]={X[i][0],X[i][1],X[i][0]+cos(a),X[i][1]+sin(a)};//.......line #1
    double B[4]={X[i-3][0],X[i-3][1],X[i-2][0],X[i-2][1]};//.............line #2
    double t[2];/*<-*/IParameters2D(t,A,B);//.....calculate intersect parameters
    Intersect2D(X[i+1],t,A);}//........use parameters to find X[i] intersections
  printf("#  x  ,   y\n");//................................print column header
  for(int i=0;i<250;++i)printf("%6.3f,%6.3f\n",X[i][0],X[i][1]);//..print points
  fclose(g);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

spiral.txt

```
#  x  ,   y
-1.000, 1.000
-1.000,-1.000
 1.000,-1.000
 1.000, 1.000
-1.000, 1.000
-0.972,-1.000
 1.000,-0.944
 0.917, 1.000
      .
      .
      .
```

NO. OF COPIES	ORGANIZATION
1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB IMAL HRA
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
24 (5 HC, 19 PDF)	DIR USARL RDRL WM P BAKER RDRL WML M ZOLTOSKI RDRL WML A M ARTHUR B BREECH P BUTLER B FLANDERS W OBERLE C PATTERSON R PEARSON L STROHM A THOMPSON R YAGER (5 HC) RDRL WML B N TRIVEDI RDRL WML C S AUBERT RDRL WML D R BEYER RDRL WML E P WEINACHT RDRL WML F D LYON RDRL WML G J SOUTH RDRL WML H J NEWILL